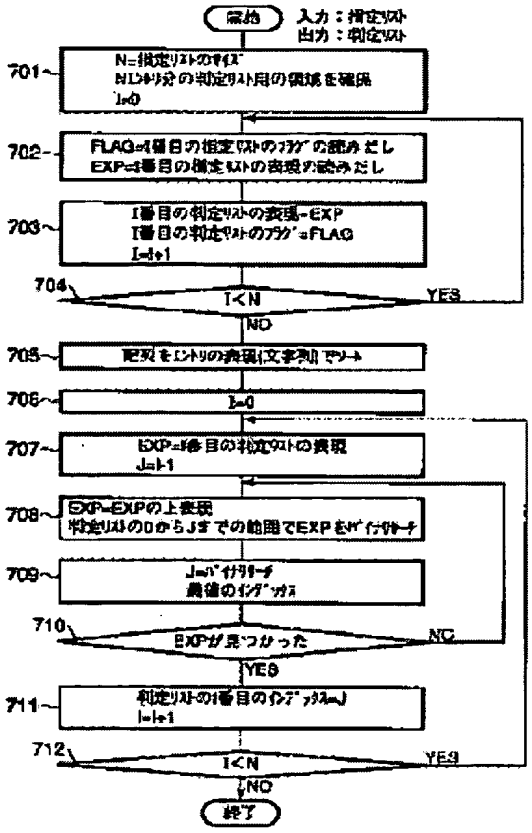
COMPILE OBJECT SPECIFYING METHOD FOR VIRTUAL MACHINE PROGRAM, COMPILE OBJECT DETERMINING METHOD FOR THE SAME, VIRTUAL MACHINE EXECUTING METHOD, PRECOMPILING METHOD, COMPUTER, STORAGE MEDIUM, AND PROGRAM PRODUCT

Patent number:	JP2002163115
Publication date:	2002-06-07
Inventor:	SAKAI RYUJI
Applicant:	TOKYO SHIBAURA ELECTRIC CO
Classification:	
- international:	G06F9/45
- european:	
Application number:	JP20000363349 20001129
Priority number(s):	JP20000363349 20001129

Report a data error here

Abstract of **JP2002163115**

PROBLEM TO BE SOLVED: To provide a method and a virtual machine capable of easily specifying and determining a compile range in an optional range including the minimum program unit when realizing the virtual machine executing an object-oriented program on a system having little memory capacity such as an embedded system. SOLUTION: The region (array) for a decision list of the entry size of a specified list is secured, the specified list is read one by one in sequence, and expression and a flag are set from the top of the decision list (701-704). The flag is set to TRUE when the character of the specified list is '+' and to FALSE when the character is '-'. The array is sorted by the expression (character string), and the index for the entry to the expression of the upper hierarchy is set for each entry (705-712).



(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2002－163115
（P2002－163115A）
(43)公開日　平成14年６月７日(2002.6.7)

(51)Int.Cl.⁷	識別記号	ＦＩ	テーマコード（参考）
Ｇ０６Ｆ　9/45		Ｇ０６Ｆ　9/44	３２２Ａ　５Ｂ０８１ ３２０Ｃ

審査請求　未請求　請求項の数19　ＯＬ　（全 14 頁）

(21)出願番号　特願2000－363349(P2000－363349)

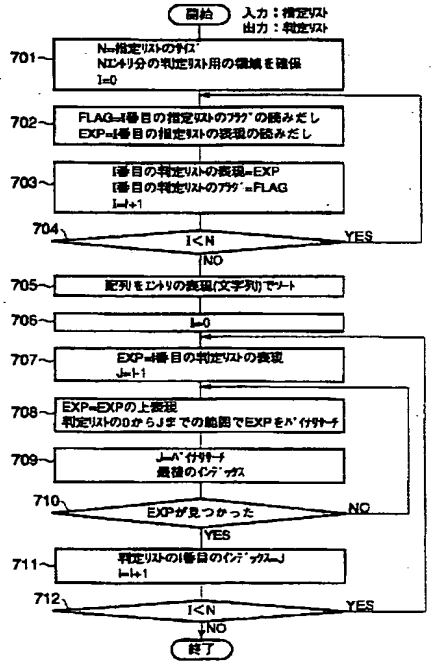
(22)出願日　平成12年11月29日(2000.11.29)

(71)出願人　000003078
株式会社東芝
東京都港区芝浦一丁目１番１号
(72)発明者　境　隆二
東京都青梅市末広町２丁目９番地　株式会社東芝青梅工場内
(74)代理人　100058479
弁理士　鈴江　武彦　（外６名）

Ｆターム(参考)　5B081 CC41 DD01

(54)【発明の名称】　仮想機械用プログラムのコンパイル対象指定方法、同プログラムのコンパイル対象判定方法、仮想機械実行方法、事前コンパイル方法、コンピュータ、記録媒体、及びプログラム製品

(57)【要約】
【課題】本発明は、組み込みシステムのようなメモリ容量の少ないシステム上に、オブジェクト指向プログラムを実行する仮想機械を実現するにあたって、コンパイル範囲を最小プログラム単位を含む任意の範囲で容易に指定し判定できる方法及び仮想機械を提供することを課題とする。
【解決手段】指定リストのエントリサイズ分の、判定リスト用の領域（配列）を確保し、指定リストを順番に逐一読み出して、判定リストの先頭から、表現とフラグを設定してゆく（７０１～７０４）。ここで、フラグは、指定リストの文字が「＋」の場合は、ＴＲＵＥを、「－」の場合はＦＡＬＳＥを設定する。次に、表現（文字列）によって、配列をソートし、各エントリについて、上位の階層の表現へのエントリへのインデックスを設定する（７０５～７１２）。



1

【特許請求の範囲】

【請求項１】　コンパイル対象となるメソッド若しくはコンパイル対象から除外したいメソッドをプログラムの階層構造を使って指定することを特徴とする仮想機械用プログラムのコンパイル対象指定方法。

【請求項２】　仮想命令コード列のインタプリタによる実行機能と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行機能とをサポートする仮想機械の実行環境に於いて、コンパイルして実行したいメソッド若しくはコンパイル対象から除外したいメソッドをプログラムのパッケージ、クラス等の階層構造を使って記述し、コンパイル対象範囲を指定することを特徴とする仮想機械用プログラムのコンパイル対象指定方法。

【請求項３】　仮想命令コード列のインタプリタによる実行機能と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行機能とをサポートする仮想機械の実行環境に於いて、コンパイルして実行したいメソッド若しくはコンパイル対象から除外したいメソッドをプログラムのパッケージ、クラス等の階層構造を使って記述したリストをもとに、特定のメソッドがコンパイル対象であるか否かを判定することを特徴とする仮想機械用プログラムのコンパイル対象判定方法。

【請求項４】　仮想命令コード列のインタプリタによる実行機能と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行機能とをサポートする仮想機械の実行環境に於いて、コンパイルして実行したいメソッド若しくはコンパイル対象から除外したいメソッドをプログラムのパッケージ、クラス等の階層構造をもとに記述したリストによりジャストインタイムコンパイルするメソッドを指定し、クラスローディング時に、当該クラスに属するメソッドをジャストインタイムコンパイルするか否かを上記リストをもとに判定してメソッド呼び出しのためのエントリに情報を設定することを特徴とする仮想機械実行方法。

【請求項５】　仮想命令コード列のインタプリタによる実行機能と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行機能とをサポートする仮想機械の実行環境に於いて、コンパイルして実行したいメソッド若しくはコンパイル対象から除外したいメソッドをプログラムのパッケージ、クラス等の階層構造をもとに記述したリストによりジャストインタイムコンパイルするメソッドを指定し、メソッド呼び出し時に、そのクラスに属するメソッドをジャストインタイムコンパイルするか否かを上記リストをもとに判定することを特徴とする仮想機械実行方法。

【請求項６】　仮想命令コード列のインタプリタによる

2

実行機能と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行機能とをサポートする仮想機械上で実行するプログラムのメソッドのうち、事前コンパイルするメソッドを、プログラムのパッケージ、クラス等の階層構造をもとに記述したリスト形式により指定し、事前コンパイル時に、前記リストを参照してコンパイルするか否かを判定することを特徴とする事前コンパイル方法。

【請求項７】　前記指定されたメソッドの事前コンパイル時に、指定されたメソッドが直接若しくは間接的に呼び出す全てのメソッドを再帰的にコンパイルすることを特徴とする請求項６記載の事前コンパイル方法。

【請求項８】　前記指定されたメソッドの事前コンパイル時に、指定されたメソッドが直接若しくは間接的に呼び出すメソッドがバーチャルメソッドであるとき、そのメソッドをオーバライドしている全てのサブクラスの対応するメソッドを再帰的にコンパイルすることを特徴とする請求項６記載の事前コンパイル方法。

【請求項９】　前記指定されたメソッドの事前コンパイル時に、指定されたメソッドが直接若しくは間接的に呼び出すメソッドがインターフェイスメソッドであるとき、そのインターフェイスクラスを実装している全てのクラスの中の対応するメソッドを再帰的にコンパイルすることを特徴とする請求項６記載の事前コンパイル方法。

【請求項１０】　仮想命令コード列のインタプリタによる実行機能と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行機能とをサポートする仮想機械に於いて、コンパイルして実行したいメソッド若しくはコンパイル対象から除外したいメソッドをプログラムのパッケージ、クラス等の階層構造を使って記述したリストを有し、前記メソッドデータに、インタプリタからのメソッド呼び出しのための第１のエントリと、目的機械の命令コード列からのメソッド呼び出しのための第２のエントリとを設けて、前記リストにより指定されたメソッドについて、前記各エントリの設定内容により、インタプリタによる実行とコンパイル済み目的機械のコード実行との実行環境の切替を制御することを特徴とする仮想機械実行方法。

【請求項１１】　前記リストにより指定されたメソッドについては、環境を変更してジャストインタイムコンパイルしてコンパイル済コードを呼び出す処理を前記第１のエントリに登録し、第２のエントリにはメソッドをジャストインタイムコンパイルしてコンパイル済コードを呼び出す処理を登録することを特徴とする請求項１０記載の仮想機械実行方法。

【請求項１２】　前記リストで指定されていないメソッドについて、前記第１のエントリに、インタプリタによる実行を続ける処理を登録することことを特徴とする請

求項１０記載の仮想機械実行方法。

【請求項１３】　前記リストで指定されたメソッドについて、前記第２のエントリには、事前コンパイル済の目的機械の命令コード列の先頭アドレスを登録し、前記第１のエントリには、環境を変更してコンパイル済コードを呼び出す処理を登録するが、再帰的にコンパイルされたメソッドについてはインタプリタによる実行を続行する処理を登録することを特徴とする請求項１０記載の仮想機械実行方法。

【請求項１４】　仮想命令コード列のインタプリタによる実行環境と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行環境とを備えたオブジェクト指向プログラムを実行する仮想機械を実現するコンピュータに於いて、

コンパイルして実行したいメソッド若しくはコンパイル対象から除外したいメソッドをプログラムのパッケージ、クラス等の階層構造を使って記述したコンパイル対象範囲を指定するリストを生成する手段と、

前記生成されたリストを参照して実行対象メソッドがコンパイル対象であるか否かを判定する手段とを具備することを特徴とするオブジェクト指向プログラムを実行する仮想機械を実現するコンピュータ。

【請求項１５】　仮想命令コード列のインタプリタによる実行環境と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行環境とを備えたオブジェクト指向プログラムを実行する仮想機械を実現するコンピュータにより読み取り可能な記録媒体であって、コンパイルして実行したいメソッド若しくはコンパイル対象から除外したいメソッドをプログラムのパッケージ、クラス等の階層構造を使って記述した、コンパイル対象範囲を指定するリストを作成する機能を実現させるためのプログラムを記録した機械読み取り可能な記録媒体。

【請求項１６】　仮想命令コード列のインタプリタによる実行環境と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行環境とを備えたオブジェクト指向プログラムを実行する仮想機械を実現するコンピュータにより読み取り可能な記録媒体であって、コンパイルして実行したいメソッド若しくはコンパイル対象から除外したいメソッドをプログラムのパッケージ、クラス等の階層構造を使って記述したリストを参照してコンパイル対象範囲を判定する機能を実現させるためのプログラムを記録した機械読み取り可能な記録媒体。

【請求項１７】　仮想命令コード列のインタプリタによる実行環境と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行環境とを備えたオブジェクト指向プログラムを実行する仮想機械を実現するコンピュータにより読み取り可能な記録媒体であって、コンパイルして実行したいメソッド若しくはコンパイル対象から除外したいメソッドをプログラムのパッケージ、クラス等の階層構造を使って記述したコンパイル対象範囲を指定するリストを作成する機能と、前記リストを参照してコンパイル対象範囲を判定する機能とを実現させるためのプログラムを記録した機械読み取り可能な記録媒体。

【請求項１８】　仮想命令コード列のインタプリタによる実行環境と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行環境とを備えたオブジェクト指向プログラムを実行する仮想機械としてコンピュータを機能させるためのプログラム製品であって、コンパイルして実行したいメソッド若しくはコンパイル対象から除外したいメソッドをプログラムのパッケージ、クラス等の階層構造を使って記述したコンパイル対象範囲を指定するリストを作成する機能をコンピュータに実現させるためのプログラム製品。

【請求項１９】　仮想命令コード列のインタプリタによる実行環境と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行環境とを備えたオブジェクト指向プログラムを実行する仮想機械としてコンピュータを機能させるためのプログラム製品であって、コンパイルして実行したいメソッド若しくはコンパイル対象から除外したいメソッドをプログラムのパッケージ、クラス等の階層構造を使って記述したコンパイル対象範囲を指定するリストを参照してコンパイル対象を判定する機能をコンピュータに実現させるためのプログラム製品。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、仮想命令コード列のインタプリタによる実行環境と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行環境とを備えたオブジェクト指向プログラムを実行する仮想機械を対象とした、仮想機械用プログラムのコンパイル対象指定方法、同プログラムのコンパイル対象判定方法、仮想機械実行方法、事前コンパイル方法、コンピュータ、記録媒体、及びプログラム製品に関する。

【０００２】

【従来の技術】仮想機械とは、コンピュータの命令セットアーキテクチャに依存しない、仮想的な命令コードによって実現されたコンピュータプログラムを、解釈実行するためのソフトウェアである。ここで、プログラムを実際に実行するコンピュータを目的機械という。

【０００３】仮想的な命令コードとしては、例えばＪＡＶＡ（登録商標）のバイトコードのように、抽象的な命令セットの場合もあるし、目的機械とは別のコンピュータの命令セットアーキテクチャの命令コードであることもある。

5

【０００４】このような仮想機械は、通常、１．プログラムの仮想命令コードを逐一読みだし、２．命令種別を判定し、３．その命令の機能を実現する処理を行う、ということを繰り返し実行する「インタプリタ」と呼ばれる方法によって実現されるが、プログラムを実行する前に、仮想命令コード列を、目的機械の命令コード列にコンパイルしてから実行する「ジャストインタイムコンパイル（またはバイナリトランスレーション）」という方法もある。

【０００５】インタプリタは、命令を解釈し実行する一つのループによって実装される。このループの中には、仮想命令をメモリからロードし命令の種類を判別して、その命令に相当する処理へ分岐する命令ディスパッチ処理と、各命令毎に、命令の機能を実行する部分があり、同等の処理を目的機械の機械語命令にコンパイルして動作するプログラムと比較すると、数倍から十数倍遅くなる。

【０００６】一方、ジャストインタイムコンパイルでは、プログラム実行時に命令列をコンパイルしなければならないため、このコンパイル処理のためにプログラムの起動時間が掛かるという問題や、コンパイラの実行のために余分にメモリ領域を必要としたり、仮想コードに比べて目的コードの方がプログラムのコードサイズが大きくなってしまうという問題がある。とくに、仮想機械のような環境を、メモリサイズに制限のある組み込みシステムのようなプラットフォームで実現しようとすると、コンパイラのためのメモリ領域や、コードサイズの増大にともなうメモリ消費の増大は大きな問題となる。

【０００７】このため、実行時にコンパイルするのではなく、事前に目的機械の命令コード列にコンパイルする「アヘッドオブタイムコンパイル」という方法によって、コンパイラ実行のための時間とコンパイラのために必要なメモリ領域を、実行時の環境から取り除くことが行われる。たたし、この場合は、仮想機械さえあれば、どのような種類の目的機械であっても、同じ仮想命令で構成されたプログラムが実行できるという、仮想機械の特性を失ってしまうことになる。

【０００８】また、一般的にプログラムの中で実行時間の多くを費しているのは、プログラムコードの極く一部であることが知られているが、この性質から、多くの時間を費している一部の部分のみを、目的機械の命令コード列に選択的にコンパイルし、残りをインタプリタで実行することによって、メモリ消費量を節約する技術もある（特開２０００－１７２５１２）。

【０００９】仮想的な命令コードを目的機械の命令コード列に選択的にコンパイルためには、選択的にコンパイルするプログラムの部分を指定するための記述が必要となるが、この際、従来では、コンパイルするべきプログラムの部分をユーザが一つずつ記述していた。ここで、選択的にコンパイルするべきプログラムの部分を指定するのは、メソッド単位で細かく指定できればよいが、それらを一つ一つ記述するのは非常に手間の掛かる作業であり、多くの時間と労力を要する。

6

【００１０】また、指定した部分が呼び出すメソッドについてもコンパイルしなければ、十分な効果が得られない。例えば、高速化したいメソッドが、システムで共通に使用するクラスライブラリの多くのメソッドを、直接または間接的に呼び出しているケースでは、これらのメソッドを全てコンパイル対象としなければ、十分な性能向上は得られない。

【００１１】さらに、このように、共通に使用するクラスライブラリのメソッドをコンパイルした場合、コンパイル対象に指定されなかったインタプリタで実行する部分から、この共通のクラスライブラリのコンパイルしたメソッドを呼び出すことになってしまう。このようなインタプリタで実行する部分から、コンパイルしたメソッドを呼び出す場合や、コンパイルした部分からインタプリタで実行する部分を呼び出す場合には、それぞれにお互いの環境を調整するための処理が必要であり、インタプリタでの実行と、コンパイル済みコードの実行を頻繁に行き来するような場合には、かえって性能が劣化するケースもある。

【００１２】

【発明が解決しようとする課題】上述したように、オブジェクト指向プログラムについて、仮想的な命令コードを目的機械の命令コード列に選択的にコンパイルためには、従来では、コンパイルするべきプログラムの部分をユーザが一つずつ記述しなければならず、従って選択的にコンパイルするべきプログラムの部分をメソッド単位で細かく指定しようとすると、その作業に非常に多くの時間と労力を要するという問題があった。また、処理性能の面から、上記指定した部分が呼び出すメソッドについてもコンパイル対象としなければ十分な性能向上は得られないが、これらの指定についても、従来では、作業が煩雑で、かつ多くの時間と労力を要するという問題があった。さらに、プログラム実行面では、共通に使用するクラスライブラリのメソッドをコンパイルした場合、インタプリタで実行する部分からコンパイルしたメソッドを呼び出す場合や、コンパイルした部分からインタプリタで実行する部分を呼び出す場合に、それぞれにお互いの環境を調整するための処理が必要であり、インタプリタでの実行と、コンパイル済みコードの実行とを頻繁に切替制御するような場合に性能劣化を招来する場合があるという問題があった。

【００１３】本発明は上記実情に鑑みなされたもので、組み込みシステムのようなメモリ容量の少ないシステム上に、オブジェクト指向プログラムを実行する仮想機械を実現するにあたって、コンパイル範囲を最小プログラム単位を含む任意の範囲で容易に指定し判定でき、これにより、仮想命令コード列のインタプリタによる実行環

境とプログラム実行時あるいは実行以前に仮想命令コード列を目的機械コード列にコンパイルして実行する実行環境をサポートする仮想機械の著しい高性能化が図れる、仮想機械用プログラムのコンパイル対象指定方法、同プログラムのコンパイル対象判定方法、仮想機械実行方法、事前コンパイル方法、コンピュータ、記録媒体、及びプログラム製品を提供することを目的とする。

【００１４】本発明は、オブジェクト指向のプログラムに於いて、目的機械の命令コード列にコンパイルする部分を最小プログラム単位を含めた任意の範囲で容易に指定することのできる仮想機械用プログラムのコンパイル対象指定方法、及びコンパイル対象を効率的に判定する仮想機械用プログラムのコンパイル対象判定方法を提供することを目的とする。

【００１５】本発明は、上記コンパイル対象指定方法を適用したプログラムの実行処理に於いて、指定したプログラム部分が呼び出すメソッドを再帰的に辿ってコンパイルすることのできる事前コンパイル方法を提供することを目的とする。

【００１６】本発明は、上記コンパイル対象指定方法を適用したプログラムの実行処理に於いて、インタプリタによる実行とコンパイル済み目的機械のコード実行とが頻繁に切り替わる実行環境での性能劣化を排除して処理の高効率化並びに高性能化を実現した仮想機械が提供できる仮想機械実行方法を提供することを目的とする。

【００１７】

【課題を解決するための手段】本発明は、仮想命令コード列のインタプリタによる実行環境と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コード列にコンパイルして実行する実行環境をサポートする仮想機械に於いて、コンパイルして実行したいメソッドを全て指定するのではなく、パッケージや、クラスの階層構造を用いて指定することにより、より少ない記述で、多くのメソッドをコンパイル対象として容易に指定できるとともに、コンパイル対象範囲外の部分を容易に除外できる仮想機械用プログラムのコンパイル対象指定方法を提供する。

【００１８】また、上記指定方法に於ける階層構造をもつデータ集合の、部分集合を定義する指定形式を使って、特定のデータが、この部分集合のメンバか否かを判定するための効率的な判定処理を実現した仮想機械用プログラムのコンパイル対象判定方法を提供する。

【００１９】また、上記指定方法を適用した際に、再帰的なコンパイルにより、コンパイル対象メソッドが呼び出す全てのメソッドを指定しなくても、高速化したい処理のメソッドのうち基のメソッドのみを指定することで、そのメソッドが直接、間接的に呼び出す全てのメソッドをコンパイルすることをコンパイル機能を実現した事前コンパイル方法を提供する。

【００２０】また、上記指定方法を適用した際に、インタプリタによる実行部分から、再帰的なコンパイルによりコンパイルされた、システム共通のクラスライブラリのメソッドを呼び出す場合は、そのままインタプリタによる実行とすることで、インタプリタによる実行からコンパイル済目的機械命令コード列の実行への制御切替えのオーバーヘッドを削減し、インタプリタによる実行とコンパイル済みコードの実行とによるハイブリッドな仮想機械の実行環境での性能劣化を排除できる仮想機械実行方法を提供する。

【００２１】

【発明の実施の形態】以下、図面を参照して本発明の実施形態を説明する。ここでは、オブジェクト指向のプログラムで、目的機械の命令コード列にコンパイルする部分を容易に指定するための実施形態を第１実施形態とし、上記コンパイル範囲指定によって、特定のメソッドがコンパイル対象となるかどうかを効率的に判定するための実施形態を第２実施形態とし、上記コンパイル範囲指定の際の指定したプログラムの部分が呼び出すメソッドを再帰的に辿ってコンパイルするための実施形態を第３実施形態とし、上記コンパイル範囲指定の際のインタプリタによる実行とコンパイル済み目的機械のコード実行を繁雑に行き来する不具合を排除するための実施形態を第４実施形態として説明する。

【００２２】１．第１実施形態（コンパイル対象となるメソッドの指定）

先ず、図１乃至図３を参照して本発明の第１実施形態について説明する。

【００２３】この第１実施形態では、仮想命令コード列のインタプリタによる実行環境と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コードにコンパイルして実行する実行環境との２つの実行環境をサポートする仮想機械において、コンパイルして実行したいメソッドを全て指定するのではなく、パッケージや、クラスの階層構造を使って、より少ない記述で、多くのメソッドを指定することを可能とし、その中からコンパイルしたくないものを容易に除外することができる指定形式を実現している。

【００２４】ＪＡＶＡのようなオブジェクト指向プログラムでは、パッケージ、クラス、メソッドのような階層構造を持っている。ここでパッケージは、その階層が存在しない場合もあるし、パッケージの下層にサブパッケージが存在し、さらにその下層にまたサブパッケージが存在するような複数階層のサブパッケージが存在する階層構造で構成される場合もある。ここでは、目的機械の命令コード列にコンパイルしたいプログラムの部分を指定するために、この階層構造を利用する。あるパッケージに含まれる全てのメソッド、あるいは、あるサブパッケージに含まれる全てのメソッド、あるいは、あるクラスに含まれる全てのメソッド、あるいは、ある特定のメソッドを表現するために、パッケージ名［．サブパッケ

ージ名・・・［．クラス名［．メソッド名］］・・・］という表現を使う。

【００２５】また、サブパッケージ名を持たない場合は、パッケージ名［．クラス名［．メソッド名］］という表現を使う。

【００２６】パッケージ名をもたない場合は、クラス名［．メソッド名］という表現を使う。

【００２７】ここで［　］は、省略可能であり、省略された場合には、その表現が示す各階層以下に含まれる全てのメソッドを意味する。

【００２８】サブパッケージの階層がより深い場合は、サブパッケージの各階層の途中までを指定することにより、そのサブパッケージに含まれる全てのメソッドを表現する。

【００２９】コンパイル対象となるメソッドの部分集合を、この表現の前に、例えば文字「＋」か「－」を付与したもののリストによって記述する。「＋」の意味は、この表現に含まれるメソッドを「コンパイル対象とする」という意味であり、「－」の意味は、この表現に含まれるメソッドを「コンパイル対象から除外する」という意味である。従って「－」に続く表現に対しては、その表現よりも上の階層の表現に「＋」が付与されたものがリストになければ、意味をなさない。また、「－」によって除外されたパッケージやクラスであっても、さらに下の階層で「＋」という記述があれば、その下の階層に含まれるメソッドはコンパイル対象となり、さらに下の階層で「－」という記述があれば、その下の階層に含まれるメソッドはコンパイル対象から除外される。このように、「＋」「－」の規則は、より下の階層による指定が優先される。

＊【００３０】この指定形式は、図１に示すように、木構造のデータ（同図（ａ）参照）に対する部分集合の指定（同図（ｂ）参照）に一般化でき、部分集合の元を逐一指定するよりも効率的である。

【００３１】上記したコンパイル対象の指定例を図２、及び図３に示す。

【００３２】図２は、ＰＡＫＡＧＥ１．ＣＬＡＳＳ１２（パッケージ１のクラス１２）に含まれるメソッドのうち「ＭＥＴＨＯＤ１２２とＭＥＴＨＯＤ１２３以外をコンパイル対象とする」という意味であり、図３は、「ＰＡＫＡＧＥ１とＰＡＫＡＧＥ２．ＣＬＡＳＳ２２以外のメソッドと、ＰＡＫＡＧＥ１．ＣＬＡＳＳ１１に含まれるメソッドのうちＭＥＴＨＯＤ１２３以外のメソッドをコンパイル対象とする」という意味である。

【００３３】このような指定をファイルに記述しておき、ジャストインタイムコンパイルや、事前コンパイルを行う処理が、このファイルを参照して、コンパイルするか否かの判定を行う。この指定は、ファイルに記述することに限定するものではなく、コマンドラインより与えることも可能であるし、プロファイラなどによって収集されたデータから自動生成され、メモリ上に記述されることもあり得る。以降、この指定を「指定リスト」と呼ぶことにする。

【００３４】この際の指定リストの記述例と、その指定リストの有用性について以下に示す。

【００３５】ここでは以下の形式の指定リストを記述したファイルを作成し、事前コンパイラの実行時オプションとしてこのファイルを指定することによって、コンパイル対象範囲を指定できるようにした。

【００３６】

```
指定リスト：
        ＤＥＳＣＲＩＰＴＩＯＮのリスト（改行で区切る）
ＥＸＰＲＥＳＳＩＯＮ：
        ＴＯＰＰＡＣＫＡＧＥＮＡＭＥ［．ＳＵＢＰＡＣＫＡＧＥＮＡＭＥ・・
・［．ＣＬＡＳＳＮＡＭＥ［．ＭＥＴＨＯＤＮＡＭＥ］］・・・］
        ＴＯＰＰＡＣＫＡＧＥＮＡＭＥ［．ＣＬＡＳＳＮＡＭＥ［．ＭＥＴＨＯ
ＤＮＡＭＥ］］
        ＣＬＡＳＳＮＡＭＥ［．ＭＥＴＨＯＤＮＡＭＥ］
ＤＥＳＣＲＩＰＴＩＯＮ：
        ＋ＥＸＰＲＥＳＳＩＯＮ
        －ＥＸＰＲＥＳＳＩＯＮ
```

上記の表現（ＥＸＰＲＥＳＳＩＯＮ）の定義で［　］が省略された場合は、その表現が示す階層以下に含まれる全てのメソッドを意味する。また、サブパッケージの階層がより深い場合には、サブパッケージの各階層の途中までの記述によって、その階層以下に含まれる全てのメソッドを意味するものとする。

【００３７】指定（ＤＥＳＣＲＩＰＴＩＯＮ）は、表現の前に「＋」または「－」を付与したものであり、「＋」の意味は、この表現に含まれるメソッドをコンパイル対象とする、という意味であり、「－」の意味は、この表現に含まれるメソッドをコンパイル対象から除外する、という意味である。

【００３８】従って、「－」に続く表現に対しては、その表現よりも上の階層の表現に「＋」が付与されたものがリストになければ、意味をなさない。また、「－」によって除外されたパッケージやクラスであっても、さらに下の階層で「＋」という記述があれば、その下の階層に含まれるメソッドはコンパイル対象となり、さらに下

の階層で「－」という記述があれば、その下の階層に含まれるメソッドはコンパイル対象から除外される。このように、「＋」「－」の規則は、より下の階層による指定が優先される。

【００３９】この指定形式は、階層構造をもつデータ集合の部分集合の指定に一般化できるので、一般化した形で、この方式が効率よく機能することを以下に説明する。

【００４０】Ｂを集合とし、Ｂの部分集合を元とする集合Ｓで、以下の性質を持つものを考える。

【００４１】任意のＳの元Ｘ，Ｙについて、Ｘ≠Ｙ　⇒　Ｘ⊂Ｙ　ＯＲ　Ｙ⊃Ｘ　ＯＲ　Ｘ∩Ｙ＝Φ

階層構造を持つデータ集合について、基底データの集合をＢ、階層構造をＳとして考えると上記の性質を持つことがわかる。

【００４２】ここで、Ｓの部分集合Ｅを指定リストに含まれる表現の集合として考えると、任意のＢの元ＢについてＢを含むＥの元が存在するなら、上記の性質Ｘ≠Ｙ　⇒Ｘ⊂Ｙ　ＯＲ　Ｙ⊃Ｘ　ＯＲ　Ｘ∩Ｙ＝Φにより、Ｂを含むＥの元の中で最小の元Ｘ∈Ｅが存在する。この場合、この表現Ｘの符号が「＋」なら、Ｂは指定した集合に含まれることになるし、符号が「－」であるか、あるいは、Ｂを含むＥの元が存在しないなら、指定した集合に含まれないことになる。このようにして、階層構造Ｓの表現を使った指定リストＥによって、集合Ｂの部分集合を効率よく指定することができる。

【００４３】上記したような記述形式による指定リストを参照して、メソッドがコンパイル対象か否かを効率よく判定することができる。この際の具体的な判定方法については後述する。

【００４４】２．第２実施形態（コンパイル対象として指定されたメソッドであるか否かの判定）

この第２実施形態では、上記した第１実施形態による指定形式を使って、特定のメソッドがコンパイル対象であるか否かを効率的に判定するコンパイル範囲判定機能を実現する。

【００４５】また、ジャストインタイムコンパイルするメソッドを上記した第１実施形態による指定形式を使って指定し、クラスローディング時に、そのクラスに属するメソッドをジャストインタイムコンパイルするか否かを上記指定リストにより判定して、メソッド呼び出しのためのエントリに情報を設定するコンパイル範囲判定機能を実現する。

【００４６】また、ジャストインタイムコンパイルするメソッドを上記した第１実施形態による指定形式を使って指定し、メソッド呼び出し時に、そのクラスに属するメソッドをジャストインタイムコンパイルするか否かを判定するコンパイル範囲判定機能を実現する。

【００４７】また、上記仮想機械上で実行するプログラム開発に於いて、開発したプログラムのメソッドの中で、事前コンパイルするメソッドを上記した第１実施形態による指定形式を使って指定し、事前コンパイル時にコンパイルするか否かを判定して、メソッドのコンパイルを制御するコンパイル範囲判定機能を実現する。

【００４８】上述した第１実施形態によるコンパイル対象範囲の指定リストを用いて、メソッドがコンパイル対象か否かを判定するアルゴリズムは、上述した指定リストの構成に於けるＢに対するＸを見つけるという手順で行う。そのために、予め上記指定リストを、表現と符号および上の階層へのポインタをメンバに持つ構造体の配列で、表現の文字列でソートした状態で内部データとして構成しておく。そして判定するメソッドについて、そのメソッドが属する「ＴＯＰＰＡＣＫＡＧＥＮＡＭＥ」から「ＭＥＴＨＯＤＮＡＭＥ」までの完全な表現を構成し、この表現をキーとして、上記の配列をバイナリサーチする。見つかれば、その表現の符号によってコンパイル対象か否かを判定する。見つからなければ、キーよりも小さいもので最も近い表現のエントリがバイナリサーチによって得られる。このエントリの表現が、キーのサブストリングであれば、そのエントリの符号によって判定し、サブストリングでなければ、そのエントリが示す上の階層へのポインタをたどって、同じことを繰り返す。上の階層へのポインタがなくなれば、そのメソッドはコンパイル対象でないと判定する。

【００４９】上述した第１実施形態に於ける指定リストを用いて、特定のメソッドがコンパイル対象であるか否かを判定する効率的な方法を示す。この方法は、指定リストから判定を行うためのデータ構造を生成する処理部分（２－１．判定リストの生成）と、特定のメソッドがコンパイル対象であるか否かを判定する処理部分（２－２．メソッドの判定処理）とでなる、２つの処理手段により実現される。ここでは、この際の判定用のデータ構造を「判定リスト」と呼ぶ。

【００５０】２－１．判定リストの生成

上記判定リストは、図４に示すように、エントリとして上記第１実施形態で示した表現（文字列）４０１と、その表現４０１がコンパイル対象であるか除外対象であるかを示すフラグ４０２と、その表現より上の階層の表現をもつエントリを指す上位階層へのインデックス４０３とでなる、データのリスト（配列）であり、第一のエントリである表現によってソートされている。この判定リストの具体例を図５と図６にそれぞれ示す。尚、図５は上記第１実施形態の図２に示した指定リストに対応する判定リスト構造を示し、図６は同じく第１実施形態の図３に示した指定リストに対応する判定リスト構造を示している。

【００５１】上記判定リストを生成する処理手順を図７にフローチャートで示す。ここで、入力は、上記指定リストであり、出力は上記判定リストである。

【００５２】先ず、指定リストのエントリサイズ分の、

判定リスト用の領域（配列）を確保し、指定リストを順番に逐一読み出して、判定リストの先頭から、表現とフラグを設定してゆく（図７ステップ７０１～７０４）。

【００５３】ここで、フラグは、指定リストの文字が「＋」の場合は、ＴＲＵＥ（真）を、「－」の場合はＦＡＬＳＥ（偽）を設定する。

【００５４】次に、表現（文字列）によって、配列をソートし、各エントリについて、上位の階層の表現へのエントリへのインデックスを設定する（図７ステップ７０５～７１２）。

【００５５】この際のインデックスの設定は、以下の手順で行う。

【００５６】先ず、該当エントリの表現から一つ上の表現を生成し、これを上表現とする。この上表現が該当エントリより小さい配列のインデックスに存在すれば、そのインデックスを設定する。存在しなければ、上表現の上表現を、新たに上表現として、同様の手順を繰り返す。空文字の上表現が見付からなかった場合は、インデックスを－１に設定する。配列は、表現でソートされているので、上表現の探索は、バイナリサーチで行うことができる。また、バイナリサーチでは、見つからなかった場合に、検索中の値の境界のインデックスが分かるので、上表現の上表現で探索し直す場合に、この０からこの境界のインデックスまでを対象とすればよい。本発明の実施形態では、バイナリサーチで見つからなかった場合の最後のインデックスは、検索中の値が、この最後のインデックスの値と、最後のインデックス－１の値の間にあるように検索する。

【００５７】２－２．メソッドの判定処理

メソッドがコンパイル対象か否かを判定する処理手順を図８にフローチャートで示す。ここでは、先ず、メソッドデータをもとに、そのパッケージ名からメソッド名までの「表現」に相当する文字列を構成し、バイナリサーチで判定リストを探索する（図８ステップ８０１～８０３）。ここで、上記「表現」に相当する文字列（ＥＸＰ）が見付かれば、見付かったエントリのフラグを判定結果として返す（図８ステップ８０４）。また上記文字列（ＥＸＰ）が見付からなければ、バイナリサーチ後の境界のインデックス－１をカレントのインデックスとする（図８ステップ８０５，８０６）。このインデックスのエントリの表現が、現在探索中の表現の上位の階層の表現である（候補（ＣＡＮＤ）が変数＝文字列（ＥＸＰ）の上位階層の表現である）か否かをチェックし、上位の表現であれば、そのエントリのフラグを返す（図８ステップ８０７，８０４）。上位の表現でなければ、そのエントリのインデックスを調べ、－１なら「コンパイルしない」フラグを返す（図８ステップ８０７～８１０）。また、－１でなければ新たにカレントインデックスとし、上記同様の処理を繰り返す（図８ステップ８０９，８０６，…）。

【００５８】上記した第２実施形態によれば、上述の第１実施形態による指定形式を使って、特定のメソッドがコンパイル対象であるか否かを効率的に判定する機能を容易に実現することができる。また、ジャストインタイムコンパイルするメソッドを上記した第１実施形態による指定形式を使って指定し、クラスローディング時に、そのクラスに属するメソッドをジャストインタイムコンパイルするか否かを上記指定リストにより判定することで、メソッド呼び出しのためのエントリに情報を設定する機能を容易に実現することができる。また、ジャストインタイムコンパイルするメソッドを上記した第１実施形態による指定形式を使って指定し、メソッド呼び出し時に、そのクラスに属するメソッドをジャストインタイムコンパイルするか否かを判定する機能を容易に実現することができる。また、上記仮想機械上で実行するプログラム開発に於いて、開発したプログラムのメソッドの中で、事前コンパイルするメソッドを上記した第１実施形態による指定形式を使って指定し、事前コンパイル時にコンパイルするか否かを判定して、メソッドのコンパイルを制御する機能を容易に実現することができる。

【００５９】３．第３実施形態（指定したメソッドからの再帰的なコンパイル）

この第３実施形態では、上記仮想機械上で実行するプログラム開発に於いて、指定されたメソッドの事前コンパイル時に、指定されたメソッドが直接、または間接的に呼び出す全てのメソッドを再帰的にコンパイルする事前コンパイル機能を実現する。

【００６０】また、上記仮想機械上で実行するプログラム開発に於いて、指定されたメソッドの事前コンパイル時に、指定されたメソッドが直接、または間接的に呼び出すメソッドがバーチャルメソッドである場合、そのメソッドをオーバライドしている全てのサブクラスの対応するメソッドを再帰的にコンパイルする事前コンパイル機能を実現する。

【００６１】また、上記仮想機械上で実行するプログラム開発に於いて、指定されたメソッドの事前コンパイル時に、指定されたメソッドが直接、または間接的に呼び出すメソッドがインターフェイスメソッドである場合、そのインターフェイスクラスを実装している全てのクラスの中の対応するメソッドを再帰的にコンパイルする事前コンパイル機能を実現する。

【００６２】また、上記仮想機械の実行環境でジャストインタイムコンパイルによって高速化する実行環境において、メソッドデータに、インタプリタからのメソッド呼び出しのためのエントリと、目的機械の命令コード列からのメソッド呼び出しのためのエントリの２つを設けて、上述した第１実施形態による指定方法によって指定されたメソッドについては、環境を変更しジャストインタイムコンパイルしてコンパイル済みコードを呼び出す処理を、インタプリタからのメソッド呼び出しのための

エントリへ登録し、目的機械の命令コード列からのメソッド呼び出しのためのエントリについては、メソッドをジャストインタイムコンパイルしてコンパイル済コードを呼び出す処理を登録する再帰的なコンパイル機能を実現する。

【００６３】特定のメソッドを高速化したい場合に、そのメソッドと、そのメソッドが呼び出す全てのメソッドを、上記の指定リストに登録する手間を省くために、高速化したいメソッドのみを指定リストに登録し、そのメソッドが呼び出すメソッドは、自動的にコンパイルしてしまうことにすれば、呼び出すメソッド全てを指定リストに登録しなくてもよい。図９に示すように、ジャストインタイムコンパイルで再帰的にコンパイルするのは、コンパイル済みメソッドが呼び出すメソッドを次々と、ジャストインタイムコンパイルして行けばよい。指定リストに含まれるメソッドをコンパイルするのは、そのクラスローディング時に、そのクラスに属するメソッドのそれぞれについて、上述した第２実施形態に示した判定手段を用いて、コンパイルするメソッドであるか否かを判定し、コンパイルするメソッドである場合には、インタプリタからの呼び出しのためのエントリに、ジャストインタイムコンパイルしてコンパイルしたメソッド実行に移る処理を登録しておき、そうでない場合は、インタプリタでの実行を続行するための処理を登録しておく。この際の実行時のメソッド呼び出しの方法については後述する第４実施形態で詳しく説明する。

【００６４】この際のメソッドを事前コンパイルする処理手順の一例を図９と、図１０乃至図１４のフローチャートに示す。事前コンパイルで指定メソッドから再帰的にコンパイルする際は、図９に示すように、先ず指定リストでコンパイル対象に指定された全てのメソッドを事前コンパイルし（図９ステップ９ａ）、続いてその各メソッドから呼ばれる全てのメソッドについて事前コンパイルする（図９ステップ９ｂ）。

【００６５】事前コンパイルで再帰的にコンパイルするのは、スタティクメソッド１０ａの場合は、呼び出されるメソッドが確定できるので、そのメソッドをコンパイルすればよい（図１１、図１４）。バーチャルメソッドや、インターフェイスメソッドなどのメソッド呼び出しでは、呼び出されるメソッドの実体が、実行時にしか断定できない。バーチャルメソッド１０ｂ呼び出しについては、実際に呼び出されるメソッドの実体は、そのメソッドのオブジェクトの宣言のクラスの対応するメソッドと、そのサブクラスによってオーバライドされたメソッドの可能性があるので、これらの全てをコンパイルする（図１２、図１４）。またインターフェイスメソッド１０ｃの場合は、そのインターフェイスをインプリメントする全てのクラスの対応するメソッドをコンパイルする（図１３、図１４）。

【００６６】上記した第３実施形態によれば、指定されたメソッドの事前コンパイル時に、指定されたメソッドが直接、または間接的に呼び出す全てのメソッドを再帰的にコンパイルする事前コンパイル機能を実現することができる。また、指定されたメソッドの事前コンパイル時に、指定されたメソッドが直接、または間接的に呼び出すメソッドがバーチャルメソッドである場合に、そのメソッドをオーバライドしている全てのサブクラスの対応するメソッドを再帰的にコンパイルする事前コンパイル機能を実現することができる。また、指定されたメソッドの事前コンパイル時に、指定されたメソッドが直接、または間接的に呼び出すメソッドがインターフェイスメソッドである場合、そのインターフェイスクラスを実装している全てのクラスの中の対応するメソッドを再帰的にコンパイルする事前コンパイル機能を実現することができる。

【００６７】４．第４実施形態（インタプリタの実行とコンパイル済みコードの実行の頻繁な切替抑止）

この第４実施形態では、仮想機械の実行環境でジャストインタイムコンパイルによって高速化する実行環境に於いて、メソッドデータに、インタプリタからのメソッド呼び出しのためのエントリと、目的機械の命令コード列からのメソッド呼び出しのためのエントリとでなる２つのエントリを設けて、上述した第１実施形態による指定方法によって指定されたメソッドについては、環境を変更しジャストインタイムコンパイルしてコンパイル済みコードを呼び出す処理を、インタプリタからのメソッド呼び出しのためのエントリへ登録し、目的機械の命令コード列からのメソッド呼び出しのためのエントリについては、メソッドをジャストインタイムコンパイルしてコンパイル済コードを呼び出す処理を登録する、再帰的なコンパイル機能を実現する。

【００６８】また上記実行環境に於いて、上記２つのエントリに対して、上述した第１実施形態による指定方法によって指定されていないメソッドについて、インタプリタからのメソッド呼び出しのためのエントリに、インタプリタによる実行を続ける処理を登録することによって、システム共通のクラスライブラリのメソッドが、再帰的にコンパイルされてしまった場合でも、インタプリタによる実行からコンパイル済み目的機械命令コード列の実行への制御切替えのオーバーヘッドを削減し、インタプリタによる実行とコンパイル済コードの実行というハイブリッドな仮想機械の実行環境での性能の劣化を回避する機能を実現する。

【００６９】また上記実行環境に於いて、上記２つのエントリに対して、上述した第１実施形態による指定方法によって指定されたメソッドについては、目的機械の命令コード列からのメソッド呼び出しのためのエントリに、事前コンパイル済の目的機械の命令コード列の先頭アドレスを登録し、インタプリタからの呼び出しのためのエントリには、環境を変更してコンパイル済コードを

呼び出す処理を登録するが、再帰的にコンパイルされたメソッドについては、インタプリタからの呼び出しのためのエントリに、インタプリタによる実行を続行する処理を登録することによって、再帰的にコンパイルされてしまった場合であってもインタプリタによる実行からコンパイル済み目的機械命令コード列の実行への制御切替えのオーバーヘッドを削減し、インタプリタによる実行とコンパイル済コードの実行というハイブリッドな仮想機械の実行環境での性能の劣化を回避する機能を実現する。

【００７０】インタプリタ実行とコンパイル済みコードの実行とが頻繁に切り替わる不具合を解消するために、ここでは、上述した第１実施形態により取得される指定リストによって指定されたメソッドがインタプリタから呼び出されるときは、コンパイル済み目的機械のコード実行へ移ってもよいが、再帰的にコンパイルされたメソッドをインタプリタから呼び出す場合は、そのままインタプリタによる実行にすることで、インタプリタ実行とコンパイル済みコードの実行の行き来（切替頻度）を削減する。

【００７１】この機能を実現するために、図１５に示すように、クラス（Ｃｉ）に含まれるメソッドデータ（ｍｉ）に、インタプリタによる実行からメソッドを呼び出すためのエントリを保持する領域（ＥＡ１）と、コンパイル済みコードの実行からメソッドを呼び出すためのエントリを保持する領域（ＥＡ２）とでなる２つのエントリ保持領域を設ける。この２つのエントリ保持領域（ＥＡ１，ＥＡ２）に於ける登録内容の一例を図１６に示す。

【００７２】ジャストインタイムコンパイラの場合は、クラスローディング時に、クラスに含まれるメソッドについて、上述の第２実施形態で説明した判定方法を使い、コンパイル対象の場合は、インタプリタからの呼び出しのためのエントリを保持する領域（ＥＡ１）に、メソッドをジャストインコンパイルしてコンパイルしたコードの実行へ移るための処理を登録する。このエントリへ登録される処理には、インタプリタからの環境（アーギュメントスタックなど）をコンパイル済みコードの実行のための環境に変換したり、メソッドがコンパイル済であれば、コンパイルせずにコンパイル済みコードを取り出すという処理が含まれる。また、コンパイル済みコードからの呼び出しのためのエントリを保持する領域（ＥＡ２）に、上記の処理から環境を変換する処理を省いたものを登録しておき、一度呼び出されると、コンパイルされるので、コンパイルされた目的機械のコード列の先頭アドレスを登録しなおす。

【００７３】コンパイル対象で無い場合は、インタプリタからの呼び出しのためのエントリを保持する領域（ＥＡ１）に、インタプリタによる実行を続行するための処理を登録し、コンパイル済みコードからの呼び出しのためのエントリを保持する領域（ＥＡ２）に、コンパイル対象の場合と同じ処理を登録する。

【００７４】以上により、コンパイル済みコードから、コンパイル対象として指定されなかったメソッドを呼ぶ場合においても、ジャストインタイムコンパイルされ、再帰的に呼び出されるメソッドがコンパイルされる。一方、インタプリタによる実行から呼び出される場合は、コンパイル対象として指定された場合には、コンパイル済み目的コードの実行になるが、そうでない場合は、インタプリタによる実行となる。

【００７５】事前コンパイルの場合には、図９に示すフローチャートの２つのループ（９ａ，９ｂ）について、最初のループ（９ａ）でコンパイルされるのが、指定リストによってコンパイル対象となったメソッドであるから、このメソッドデータのインタプリタからの呼び出しのためのエントリを保持する領域（ＥＡ１）に、環境を変更してコンパイル済みコードの先頭アドレスを呼び出す、一連の処理を登録し、コンパイル済み目的コードの実行からの呼び出しのためのエントリを保持する領域（ＥＡ２）に、事前コンパイル済みの目的コード列の先頭アドレスを登録する。第２のループ（９ｂ）によってコンパイルされるメソッドについては、インタプリタからの呼び出しのためのエントリを保持する領域（ＥＡ１）に、インタプリタによる実行をそのまま続けるための処理を登録し、コンパイル済みコードからの呼び出しのためのエントリを保持する領域（ＥＡ２）に、事前コンパイル済の目的コード列の先頭アドレスを登録する。どちらのループでもコンパイルされないメソッドについては、インタプリタからの呼び出しのためのエントリには、インタプリタによる実行をそのまま続けるための処理を登録し、コンパイル済コードからの呼び出しのためのエントリには、インタプリタによる実行へ戻るための処理を登録する。この最後の設定は、再帰的なコンパイルを抑制したいようなメソッドについて適用可能であり、その場合はジャストインタイムコンパイルのケースも同様である。

【００７６】このようにして、インタプリタ実行とコンパイル済みコードの実行の頻繁な切替を抑止することにより、インタプリタによる実行からコンパイル済み目的機械命令コード列の実行への制御切替えのオーバーヘッドを削減し、インタプリタによる実行とコンパイル済コードの実行というハイブリッドな仮想機械の実行環境での性能の劣化を防ぐことができる。

【００７７】尚、上記した各実施形態に於ける処理プログラム並びに各処理の成果物（データ）等は、それぞれ個別に若しくは合体して、例えば、目的機械を構成するコンピュータの内部メモリに格納される。若しくは、ハードディスク装置、フロッピィディスク、メモリカード等の各種外部記憶装置等に記憶し、若しくはデータ通信手段等を介して、一定の条件下で、例えばプログラム製

品、記憶媒体等として扱うことも可能である。

【００７８】

【発明の効果】以上詳記したように本発明によれば、仮想命令コード列のインタプリタによる実行環境と、プログラム実行時あるいは実行以前に仮想命令コード列を目的機械コード列にコンパイルして実行する実行環境とによる２つの実行環境をサポートする仮想機械に於いて、コンパイルして実行したいメソッドを全て指定するのではなく、パッケージや、クラスの階層構造を使って、より少ない記述で、多くのメソッドを指定することを可能とし、コンパイルしたくないプログラム部分を容易に除外することができる。

【００７９】また、上記階層構造をもつデータ集合の、部分集合を定義する指定形式を使って、特定のデータが、この部分集合のメンバであるか否かを判定するための効率的な判定処理機能を実現できる。

【００８０】また、再帰的なコンパイルにより、コンパイル対象メソッドが呼び出す全てのメソッドを指定しなくても、高速化したい処理のメソッドの基のメソッドのみを指定することで、そのメソッドが直接、間接的に呼び出す全てのメソッドをコンパイルすることができる。

【００８１】また、インタプリタによる実行部分から、再帰的なコンパイルによりコンパイルされた、システム共通のクラスライブラリのメソッドを呼び出す場合は、そのままインタプリタによる実行とすることで、インタプリタによる実行からコンパイル済目的機械命令コード列の実行への制御切替えのオーバーヘッドを削減し、インタプリタによる実行とコンパイル済コードの実行というハイブリッドな仮想機械の実行環境での性能の劣化を防ぐことができる。

【図面の簡単な説明】

【図１】本発明の第１実施形態に係るプログラム階層構造を用いたコンパイル対象範囲の指定方法を説明するための、階層構造（木構造）とそのデータ集合の領域を示す図。

【図２】上記第１実施形態に於けるコンパイル対象の一指定例を示す図。

【図３】上記第１実施形態に於けるコンパイル対象の他の指定例を示す図。

＊【図４】本発明の第２実施形態に係るコンパイル対象判定方法を説明するためのデータ構造を示す図。

【図５】上記第２実施形態に於ける判定リストの一構成例を示す図。

【図６】上記第２実施形態に於ける判定リストの他の構成例を示す図。

【図７】上記第２実施形態に於ける判定リストを生成する処理手順を示すフローチャート。

【図８】上記第２実施形態に於けるコンパイル対象を判定する処理手順を示すフローチャート。

【図９】本発明の第３実施形態に係る、事前コンパイルで指定メソッドから再帰的にコンパイルする処理手順を示すフローチャート。

【図１０】上記第３実施形態に於けるメッセージの事前コンパイル処理手順を示すフローチャート。

【図１１】上記第３実施形態に於けるメッセージの事前コンパイル処理手順を示すフローチャート。

【図１２】上記第３実施形態に於けるメッセージの事前コンパイル処理手順を示すフローチャート。

【図１３】上記第３実施形態に於けるメッセージの事前コンパイル処理手順を示すフローチャート。

【図１４】本発明の第４実施形態に係るメソッドデータの構造を示す図。

【図１５】上記第４実施形態に於ける、メソッドデータに設けられたインタプリタによる実行からメソッドを呼び出すためのエントリを保持する領域（ＥＡ１）と、コンパイル済みコードの実行からメソッドを呼び出すためのエントリを保持する領域（ＥＡ２）を示す図。

【図１６】上記第４実施形態に於ける２つのエントリ保持領域（ＥＡ１，ＥＡ２）の登録内容を示す図。

【符号の説明】

４０１…エントリデータの表現（文字列）

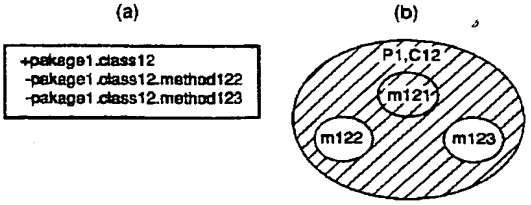
４０２…エントリデータのフラグ

４０３…上位階層へのインデックス

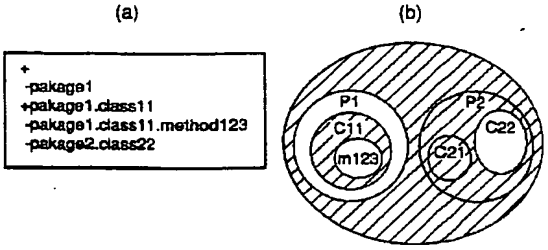
ＥＡ１…インタプリタによる実行からメソッドを呼び出すためのエントリを保持する領域

ＥＡ２…コンパイル済みコードの実行からメソッドを呼び出すためのエントリを保持する領域

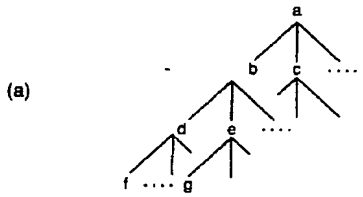
【図２】



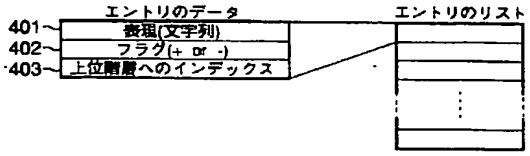
【図３】



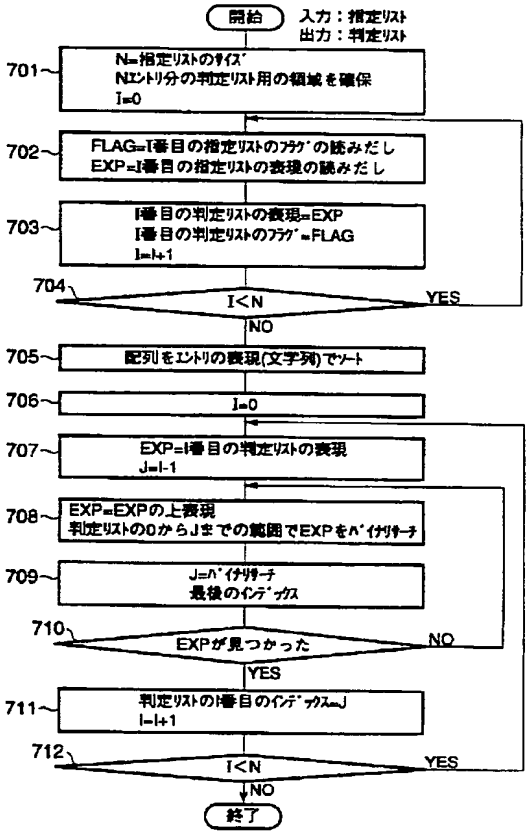
【図１】



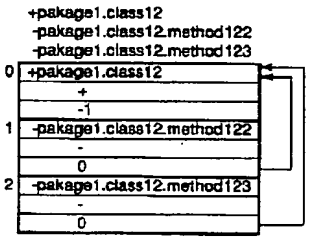
【図４】



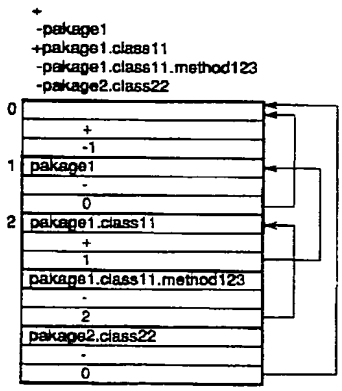
【図７】



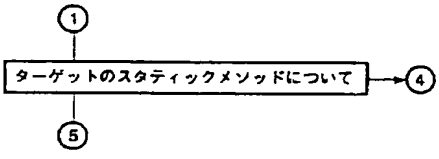
【図５】



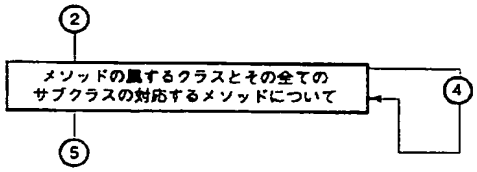
【図６】



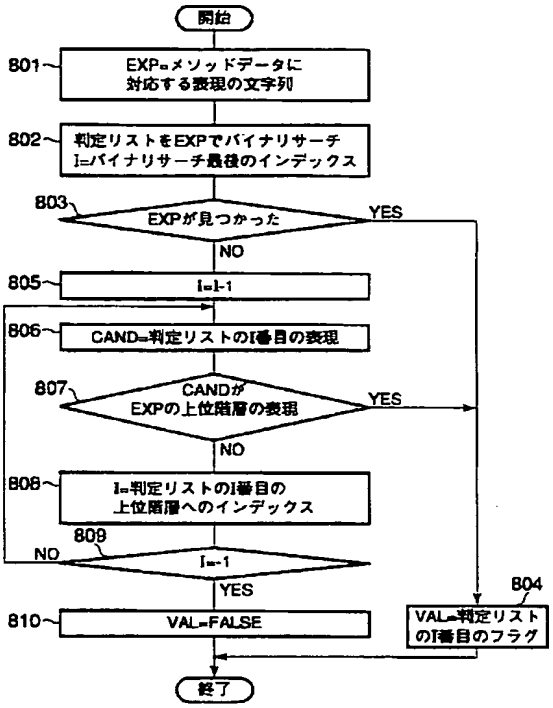
【図１１】



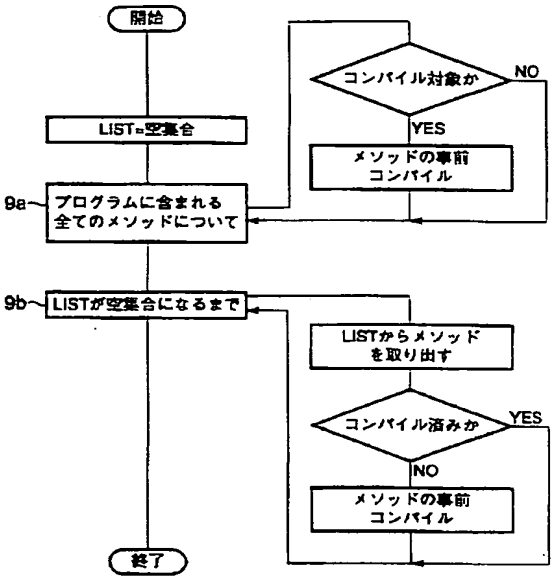
【図１２】



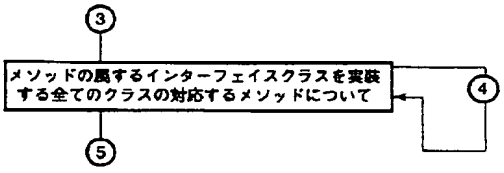
【図８】



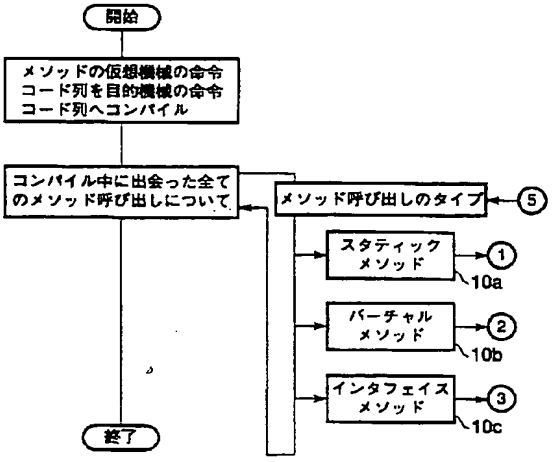
【図９】



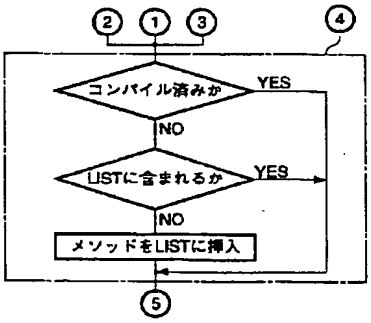
【図１３】



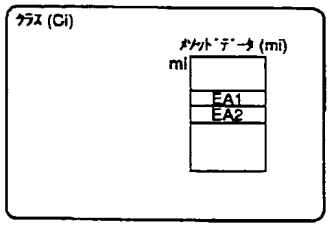
【図１０】



【図１４】



【図１５】



【図１６】

条件/対象 / エントリ	指定したもの	リカーシブにコンパイルされたライブラリなど	その他
EA1	コンパイル済コードを呼出すためのエントリ	インタプリタの実行	インタプリタの実行
EA2	コンパイル済コードのエントリアドレス	コンパイル済コードのエントリアドレス	インタプリタの実行に戻るための処理